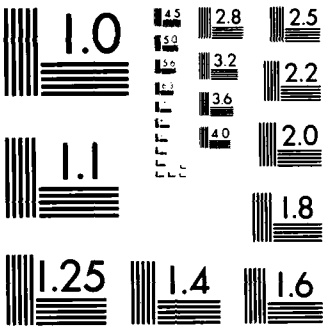
AD-A156 722 ALTERNATING DIRECTION METHODS ON MULTIPROCESSORS: AN EXTENDED ABSTRACT(U) YALE UNIV NEW HAVEN CT DEPT OF COMPUTER SCIENCE Y SAAD ET AL. APR 85 YALEU/DCS/RR-381 1/1

UNCLASSIFIED N00014-82-K-0184 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

Alternating Direction Methods on Multiprocessors : An Extended Abstract

Youcef Saad and Martin H. Schultz

Research Report YALEU/DCS/RR-381
April 1985

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Abstract. We propose a few implementations of the Alternating Direction Method for solving parabolic partial differential equations on multiprocessors. A careful complexity analysis of these implementations shows that, contrary to what is generally believed, the method can be made highly efficient on parallel architectures by using pipelining and variations of the classical Gaussian elimination algorithm for solving tridiagonal systems.

Alternating Direction Methods on Multiprocessors : An Extended Abstract

Youcef Saad and Martin H. Schultz

Research Report YALEU/DCS/RR-381
April 1985

This work was supported in part by ONR grant N00014-82-K-0184 and in part by a joint study with IBM/Kingston.

1. Introduction

We propose a few implementations of the Alternating Direction Method for solving parabolic partial differential equations on multiprocessors. A careful complexity analysis of these implementations shows that, contrary to what is generally believed, the method can be made highly efficient on parallel architectures by using pipelining and variations of the classical Gaussian elimination algorithm for solving tridiagonal systems.

2. The Alternating Direction Method and its parallel implementations

We consider the partial differential equation:

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(a(x,y)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(b(x,y)\frac{\partial u}{\partial y}\right)$$

on the domain $(x,y,t) \in \Omega \times [0,T] \equiv [0,1] \times [0,1] \times [0,T]$, with the initial and boundary conditions:

$$u(x,y,0) = u_0(x,y), \quad \forall (x,y) \in \Omega,$$
$$u(\bar{x},\bar{y},t) = g(\bar{x},\bar{y},t), \quad \forall (\bar{x},\bar{y}) \in \partial\Omega, \quad t>0,$$

where $\partial\Omega$ is the boundary of the unit square Ω.

A common approach to solve the above problem is the Alternating Direction Method. First the equations are discretized with respect to the space variables x and y using a mesh of $n+1$ points in each direction. The result is the system of $N \equiv n^2$ ordinary differential equations:

$$\frac{d\underline{u}}{dt} = A_x\underline{u} + B_y\underline{u} \tag{2.1}$$

in which the matrices A_x and B_y represent the 3-point central difference approximations to the operators $\frac{\partial}{\partial x}(a(x,y)\frac{\partial}{\partial x})$ and $\frac{\partial}{\partial y}(b(x,y)\frac{\partial}{\partial y})$ respectively.

The Alternating Direction Method algorithm consists in stepping (2.1) forward in time alternately in the x and y directions as follows [4]:

$$(I - \frac{1}{2}\Delta t A_x)u^{i+\frac{1}{2}} = (I + \frac{1}{2}\Delta t B_y)u^i \tag{2.2}$$

$$(I - \frac{1}{2}\Delta t B_y)u^{i+1} = (I + \frac{1}{2}\Delta t A_x)u^{i+\frac{1}{2}}. \tag{2.3}$$

We observe that if the mesh points are ordered by lines in the x direction, then (2.2) constitutes a set of n independent tridiagonal systems whose solution is perfectly parallellizable. However, (2.3) constitutes one tridiagonal system in which the nonzero diagonals are the main diagonal and the $(n+1)^{st}$ super- and sub-diagonals. It is important to note that this second system can also be recast into a set of n independent tridiagonal systems by *reordering the grid points* by lines, this time in the y direction. This essentially amounts to transposing the matrix representing the solution at the $n \times n$ grid points and is an expensive data permutation operation which is often cited as the main drawback of Alternating Direction Method in regard to its implementation on parallel machines. The other difficulty that has been traditionally associated with parallelizing the Alternating Direction Method is that the classical algorithms for solving tridiagonal systems are sequential in nature. Various parallel implementations of the Alternate Direction Method have been discussed by Gannon and Van Rosendale [1] Lambiotte [2] and Ortega and Voigt [3].

Since it is the tridiagonal systems that are usually troublesome, we will consider the costs of only the two tridiagonal system solutions in (2.2) and (2.3). On a single processor each half step costs $5n^2$ operations for the forward and $3n^2$ operations for the backward sweep. Assuming that s arithmetic operations can be done per second, the time for a half step on a single processor is approximately

$$T_1 = \frac{8n^2}{s}. \tag{2.4}$$

3. Alternating Direction Methods on the shared memory model and the broadcast bus model

In this section, we assume that k processors are connected to one large shared memory with total bandwidth B words per second and start up τ. Moreover, we assume that the I/O bandwidth of each processor is b. Thus B/b different processors can simultaneously access the shared memory.

The transfer of data between two processors is done by the first processor writing to the shared memory and then the second processor reading from the shared memory. Consider a method consisting in solving n/k similar tridiagonal systems in each processor for (2.2) then transposing the data and solving (2.3) as n/k tridiagonal systems in each processor (it is assumed that $k \leq n$). The total time required for performing the arithmetic operations of one half step is approximately

$$T_{S,Arith} \approx \frac{n}{k}\frac{8n}{s} = \frac{8n^2}{ks}$$

Performing a transpose of the n x n matrix of the unknown u requires the communication time:

$$\begin{aligned} T_{S,Comm} &\approx 2\frac{k}{B/b}\left(\tau + \frac{n^2}{kb}\right) \\ &\approx 2k\frac{b}{B}\tau + 2\frac{n^2}{B}. \end{aligned}$$

Hence in the shared memory model, it takes a total time of

$$T_S \approx 2k\frac{b}{B}\tau + 2\frac{n^2}{B} + \frac{8n^2}{ks}$$

to perform one half step of the Alternating Direction Method.

Assume now that all k processors are linked to each other via a global bus which allows for broadcasting. The time to perform the arithmetic operations is the same as for the shared memory model.

To transpose the matrix, each processor broadcasts in turn its data to all the other processors. This requires a total communication time of

$$T_{B,Comm} \approx k(\tau + \frac{n^2}{kb}) = k\tau + \frac{n^2}{b}$$

Hence the total time

$$T_B \approx k\tau + \frac{n^2}{b} + \frac{8n^2}{ks}$$

for performing one half step of Alternating Direction Method.

1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

Figure 1: Domain decomposition and assignment of the unit square into 4 processors.

Observe that neither of the above two models achieves a reasonable speed up when k increases because of the limiting term n^2 in the communcation cost. This is the source of the belief that Alternating Direction Methods do not parallelize well.

4. Alternating Direction Methods on a multiprocessor ring

In the ring architecture the k processors are arranged in a ring and each processor can communicate with its two nearest neighbors. It is assumed throughout that the processors are numbered so that processor numbered i is connected to processors nubered $i+1$ and $i-1$ modulo k. In order to efficiently implement the algorithm, we will first assign the grid points such as to minimize data communication costs. The assignment, which is described in Figure 1 for a ring of $k = 4$ processors, will essentially avoid transposing the data at each half step.

This assignment results in each of the tridiagonal systems (2.2), being split into k equal parts, one part per processor. *We consider the cost of the half step involving the sweep in the x-direction.* Looking at the leftmost column of subsquares in Figure 1, each of the k processors of that column can start performing the forward sweeps simultaneously on the n/k tridiagonal systems that it holds. Thus, the sweep eliminates the variables horizontally from left to right. When the boundary of the first subsquare is reached, each processor sends to its right neighbor in the figure the data that is necessary for that processor to continue its forward sweep on the second part of the tridiagonal matrix, i.e., the processor numbered j sends to its neighbor numbered $(j+1)$ Mod k the $(j-1)\frac{n}{k}+1$ to $j\frac{n}{k}$ rows of the tridiagonal system. The forward sweep can now be pursued on the second column of subsquares, and this process is repeated until the forward sweep is completed on the variables of the last column of subsquares. The backward sweep is accomplished in a similar fashion except that we proceed from right to left. The half step in the y direction is performed similarly, with the forward sweep proceeding from bottom to top and the backward sweep from top to bottom. We observe that no processor is idle at any time. It is clear that the time to perform the arithmetic operations is again of the form:

$$T_{R,Arith} \approx \frac{n}{k}\frac{8n}{s} = \frac{8n^2}{ks}.$$

When we cross an interface in the forward sweep, we must transfer the pivot row plus the corresponding element of the right hand side of each of the n tridiagonal systems in each processor numbered j to the processor numbered $(j+1)$ Mod k. These I/O operations can be done in parallel for each of the processors on the same column. A total of $k-1$ interfaces must be crossed. For the backward sweep, we transfer only one element of the current solution per system. Therefore:

$$T_{R,Comm} = (k-1)\left(\tau + \frac{3n}{kb}\right) + (k-1)\left(\tau + \frac{n}{kb}\right) \approx 2(k-1)\tau + \frac{4n}{b}.$$

Thus, the total execution time comes to

$$T_R(k) = 2(k-1)\tau + \frac{4n}{b} + \frac{8n^2}{ks}.$$

Clearly, the time for solving in the other direction is identical.

We observe that with increasing k the arithmetic time decreases while the communication time increases linearly. The function $T_R(k)$ has a minimum which is achieved for

$$k_{opt} \approx \frac{2n}{\sqrt{s\tau}}$$

and the corresponding optimal time is linear in n:

$$T_{R,opt} \approx 4\left(2\sqrt{\frac{\tau}{s}} + \frac{1}{b}\right)n.$$

Thus by an appropriate choice of algorithm on the ring, with $O(n)$ processors we can achieve an $O(n)$ speed-up contradicting the conventional wisdom.

References

[1] D. Gannon, J. van Rosendale, *On the Impact of Communication Complexity in the Design of Parallel Algorithms,* Technical Report 84-41, ICASE, 1984.

[2] J.J. Lambiotte, *The solution of linear systems of equations on a vector computer,* Ph.D. Thesis, University of Virginia, 1975.

[3] J.M. Ortega, R.G. Voigt, *Solution of partial differential equations on vector and parallel computers,* Technical Report 85-1, ICASE, NASA Langley Research Center, 1985.

[4] R.S. Varga, *Matrix Iterative Analysis,* Prentice Hall, Englewood Cliffs, New Jersey, 1962.

END

FILMED

8-85

DTIC